DTIC FILE COPY

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

AD-A223 764

1. AGEN E

3. REPORT TYPE AND DATES COVERED
Final 28 Nov 89 to 28 Nov 90

4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report:Apollo Computer Inc., Domain ADA, Version 3.0 MBX, DN 4000 (Host) to MVME 133A-20 (Target), 891128S1.10234

5. FUNDING NUMBERS

6. AUTHOR(S)
National Institute of Standards and Technology
Gaithersburg, MD
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Institute of Standards and Technology
National Computer Systems Laboratory
Bldg. 255, Rm. A266
Gaithersburg, MD 20899
USA

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Apollo Computer Inc., Domain ADA, Version 3.0 MBX, Gaithersburg MD, DN 4000 under Domain /OS SR 10.2 (Host) to MVE 133A-20 under N/A (Target), ACVC 1.10.

14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

AVF Control Number: NIST89APO585_1.10
23 January 1990
DATE VSR MODIFIED PER AVO COMMENTS: 04-30-90

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 891128S1.10234
Apollo Computer Inc
Domain ADA, Ver 3.0.MBX
DN 4000 Host and MVME 133A-20 Target

Completion of On-Site Testing:
28 November 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: Domain/ADA

Certificate Number: 891128S1.10234

Host: DN 4000 under Domain/OS SR 10.2

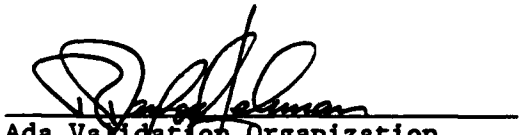
Target: MVE 133A-20 under N/A

Testing Completed 28 November 1989 Using ACVC 1.10

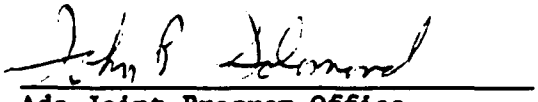
This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899



for Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

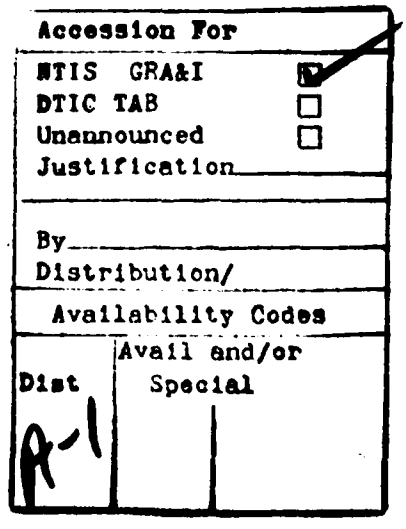


TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by Gemma Corp under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 28 November 1989 at Apollo.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure

consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved

words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity

functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Domain/ADA, Ver 3.0.MBX

ACVC Version: 1.10

Certificate Number: 891128S1.10234

Host Computer:

Machine: DN 4000

Operating System: Domain/OS SR 10.2

Memory Size: 12 MB

Target Computer:

Machine: MVME 133A-20

Operating System: N/A

Memory Size: 1 MB

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, TINY_INTEGER, and SHORT_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) CONSTRAINT_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the

range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round to even. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

(2) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array objects are declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR

either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

(4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(7) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(8) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

j. Input and output.

(1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)

(5) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(6) RESET and DELETE operations are not supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(7) RESET and DELETE operations are not supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(8) RESET and DELETE operations are not supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(9) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(10) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)

(11) Temporary direct files are given names and deleted when closed. (See test CE2108C.)

(12) Temporary text files are given names and deleted when closed. (See test CE3112A.)

(13) More than one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(14) More than one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(15) More than one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A..B, (2 tests), CE3111D..E (2 tests), and CE3114B.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 326 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 10 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	129	1133	1994	17	28	46	3347
Inapplicable	0	5	321	0	0	0	326
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	198	577	545	245	172	99	162	331	137	36	252	294	299	3347
Inapplicable	14	72	135	3	0	0	4	1	0	0	0	75	22	326
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B	E28005C	ED7004B	ED7005C	ED7005D
ED7006C	ED7006D				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 326 tests were inapplicable for the reasons indicated:

a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C241131..Y (14 tests) C357051..Y (14 tests)
C357061..Y (14 tests) C357071..Y (14 tests)
C357081..Y (14 tests) C358021..Z (15 tests)
C452411..Y (14 tests) C453211..Y (14 tests)
C454211..Y (14 tests) C455211..Z (15 tests)
C455241..Z (15 tests) C456211..Z (15 tests)
C456411..Y (14 tests) C460121..Z (15 tests)

b. C35702B and B86001U are not applicable because this implementation supports no predefined type LONG_FLOAT.

c. The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C C45304C C45502C C45503C C45504C
C45504F C45611C C45613C C45614C C45631C
C45632C B52004D C55B07A B55B09C B86001W
CD7101F

d. C45531M, C45531N, C45532M, and C45532N are not applicable because the fixed point type definitions

```
DELTA 2.0**-48 RANGE -0.5 .. 0.5 -2.0**-48
DELTA 2.0**-47 RANGE -1.0 .. 1.0 -2.0**-47
DELTA 2.0**-46 RANGE -2.0 .. 2.0 -2.0**-46
```

are not supported.

e. C45531O, C45531P, C45532O, and C45532P are not applicable because the fixed point type definitions

```
DELTA 0.5 RANGE -2.0**46 .. 2.0**46 -0.5
DELTA 1.0 RANGE -2.0**47 .. 2.0**47 -1.0
DELTA 2.0**46 RANGE -2.0**93 .. 2.0**93 -2.0**46
```

are not supported.

f. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

g. C86001F is not applicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. These tests recompile package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

h. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

i. The following tests are not applicable because 'SIZE representation clauses for floating-point types are not supported:

CD1009C CD2A41A CD2A41B CD2A41E
CD2A42A..J (8 tests)

j. The following tests are not applicable because 'SIZE representation clauses for derived array types are not supported:

CD2A61I CD2A61J

k. The following tests are not applicable because 'SIZE representation clauses for access types are not supported:

CD2A84B..CD2A84I (8 tests)
CD2A84K CD2A84L

l. The following tests are not applicable because 'SIZE representation clauses for task types are not supported:

CD2A91A..CD2A91E (5 tests)

m. The following tests are not applicable because 'ADDRESS clauses for variables of type SYSTEM.ADDRESS are limited in that an address clause with a dynamic address is applied to a variable requiring initialization. In addition, address clauses are not supported for tasks.

CD5003B..CD5003H (7 tests) CD5011A CD5011C
CD5011E CD5011G CD5011M CD5011Q CD5012A
CD5012B CD5012E CD5012F CD5012I CD5012J
CD5013S CD5014S CD5014T CD5014V CD5014X

n. The following tests are not applicable because 'ADDRESS clauses for constant types of type SYSTEM.ADDRESS are limited in that an address clause with a dynamic address is applied to a constant requiring initialization.

CD5011B CD5011D CD5011F CD5011H CD5011I
CD5011N CD5011R CD5012C CD5012D CD5012G
CD5012H CD5012L CD5013B CD5013D CD5013F
CD5013H CD5013L CD5013N CD5013R CD5014U
CD5014W

o. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

p. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

q. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

r. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

s. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

t. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

u. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

v. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

w. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

x. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

y. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

z. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

aa. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

ab. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

ac. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ad. CE3102E is inapplicable because text file CREATE with IN_FILE mode is supported by this implementation.

ae. CE3102F is inapplicable because text file RESET is supported by this implementation.

af. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

ag. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

ah. CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

ai. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

aj. CE3115A is inapplicable because this implementation does not support RESET to mode OUT_FILE when another internal file is associated with the same external file which is opened to mode IN_FILE.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 10 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B24009A B33301B B38003A B38003B B38009A
B38009B B41202A B91001H BC1303F BC3005B

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the Domain/ADA, Ver 3.0.MBX compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Domain/ADA, Ver 3.0.MBX compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	DN 4000
Host operating system:	Domain/OS SR 10.2

Target computer: MVME 133A-20
Target operating system: N/A

A tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precision was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were not customized before being written to the tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the tape.

TEST INFORMATION

The contents of the tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled , linked, and all executable tests were run on the DN 4000.

The compiler was tested using command scripts provided by Apollo and reviewed by the validation team. See Appendix E for a complete listing of the compiler options for this implementation. The compiler options invoked during this test were:

ADA-M (Test File Name) for single file tests

ADA (Test File Name) and A.LD (Main Test Name) for multiple part tests

Tests were compiled, linked, and executed (as appropriate) using a two identical DN 4000 computers. Test output, compilation listings, and job logs were captured on tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Apollo Computer Inc. and was completed on 28 November 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

Apollo has submitted the following Declaration of Conformance concerning the DN 4000.

APPENDIX A

Declaration of Conformance

Customer:	Apollo Computer, Inc.
Ada Validation Facility:	National Institute of Standards & Technology
ACVC Version:	1.10

Ada Implementation

Ada Compiler Name:	Domain/ADA
Version:	V3.0.mbx
Host Computer System:	DN4000 (Operating System = Domain/OS SR10.2
Target Computer System:	MVME 133A-20 (with no Operating System)

Customer's Declaration

I, the undersigned, representing <~~customer~~> Apollo Computer declare that <~~customer~~> Apollo Computer has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation(s) listed in this declaration.

Signature

November 17, 1989
Date

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the Domain/ADA, Ver 3.0.MBX compiler, as described in this Appendix, are provided by Apollo Computer Inc. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.
Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

 type INTEGER is range -2_147_483_648 ..2_147_483_647;
 type SHORT_INTEGER is range -32_768 .. 32_767;
 type TINY_INTEGER is range -128 .. 127;

 type FLOAT is digits 15 range -1.79769313486232E+308 ..
                  1.79769313486232E+308;
 type SHORT_FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

 type DURATION is delta 1.00000000000000E-03 range -2147483.647 ..
                                                    2147483.647;

...

end STANDARD;
```

Attachment I

Appendix F

Implementation–Dependent Characteristics

This appendix summarizes the implementation–dependent characteristics of the Domain/Ada Cross–Development System as required by Appendix F of the *Ada Reference Manual* (RM). In particular, this appendix

- Lists the Domain/Ada pragmas and attributes
- Gives the specification for the **package** SYSTEM
- Lists the restrictions on representation clauses and unchecked type conversions
- Gives the naming conventions for denoting implementation–dependent components in record representation clauses
- Gives the interpretations of expressions in address clauses
- Presents the implementation–dependent characteristics of I/O packages
- Presents any additional implementation–dependent features

The following sections summarize each of these topics.

F.1 Implementation–Dependent Pragmas and Attributes

This section details the implementation–dependent pragmas and attributes of Domain/Ada.

F.1.1 Implementation-Dependent Pragmas

Domain/Ada provides the following pragmas. Some of the entries in this list refer you to the chapter or section in this document where you can find additional information about the pragma.

- **pragma** BUILT_IN may be used in some parts of the code for TEXT_IO, MACHINE_CODE, UNCHECKED_CONVERSION, UNCHECKED_DEALLOCATION, and lower level support packages in STANDARD. It is reserved for use by Apollo and is not directly accessible to the user.

- **pragma** IMPLICIT_CODE specifies that implicit code generated by the compiler is allowed (ON) or disallowed (OFF). You can use this pragma only within the declarative part of a machine code procedure. (Refer to Sections 9.9.3 and 11.8 for more information.)

- **pragma** INLINE_ONLY, when used in the same way as **pragma** INLINE, indicates to the compiler that the subprogram must *always* be inlined. (This is very important for some code procedures.) This pragma also suppresses the generation of a callable version of the routine, which saves code space.

- **pragma** NO_IMAGE suppresses the generation of the image array used for the IMAGE attribute of enumeration types. This eliminates the overhead required to store the array in the executable image.

- **pragma** NON_REENTRANT takes one argument which can be the name of a library subprogram or a subprogram declared immediately within a library package spec or body. This pragma indicates to the compiler that the subprogram will not be called recursively allowing the compiler to perform specific optimizations. The pragma can be applied to a subprogram or a set of overloaded subprograms within a package spec or package body.

- **pragma** NOT_ELABORATED, which is allowed only within a package specification, suppresses elaboration checks for all entities defined within a package, including the package specification itself. In addition, this pragma suppresses the generation of elaboration code. When using **pragma** NOT_ELABORATED, you must ensure that there are no entities defined in your program that require elaboration.

- **pragma** PASSIVE has three forms:

  pragma PASSIVE
  pragma PASSIVE(SEMAPHORE):
  pragma PASSIVE(INTERRUPT, NUMBER);

  This pragma can be applied to a task or task type declared immediately within a library package spec or body. It directs the compiler to optimize certain tasking operations. It is possible that the statements in the task body will prevent the intended optimization, in these cases a warning will be generated at compile time and will raise TASKING_ERROR at run time.

- **pragma** SHARE_CODE provides for the sharing of object code between multiple instantiations of the same generic procedure or package body. A "parent" instantiation is created and subsequent instantiations of the same types can share the parent's object code, reducing program size and compilation times. You can use the name **pragma** SHARE_BODY instead of SHARE_CODE with the same effect. (Refer to Section 11.1 for more information.)

In addition to the pragmas mentioned in the previous list, Domain/Ada expands upon the functionality of the following predefined language pragmas:

- **pragma** INLINE is implemented as described in Appendix B of the RM with the addition that you can expand recursive calls up to the maximum depth of 8. The compiler produces warnings for nestings that are too deep or for bodies that are not available for inline expansion.

- **pragma** PACK causes the compiler to minimize gaps between components in the representation of composite types. For arrays, the compiler packs components to bit sizes corresponding to powers of 2 (if the field is smaller than STORAGE_UNIT bits). The compiler packs objects larger than a single STORAGE_UNIT to the nearest STORAGE_UNIT.

- **pragma** SUPPRESS is supported in the single parameter form. The pragma applies from the point of occurrence to the end of the innermost enclosing block. You cannot suppress DIVISION_CHECK. The double parameter form of the pragma with a name of an object, type, or subtype is recognized, but has no effect in the current release. (Refer to Section 9.9.2 for more information.) You can use this pragma to suppress elaboration checks on any compilation unit except a package specification.

This implementation recognizes the following pragmas, but they have no effect in the current release:

- **pragma** CONTROLLED

- **pragma** MEMORY_SIZE

- **pragma** OPTIMIZE (Refer to the **ada** -O option for code optimization in Chapter 4.)
- **pragma** SHARED
- **pragma** STORAGE_UNIT (This implementation does not allow you to modify **package** SYSTEM by means of pragmas. However, you can achieve the same effect by recompiling **package** SYSTEM with altered values.)
- **pragma** SYSTEM_NAME (This implementation does not allow you to modify **package** SYSTEM by means of pragmas. However, you can copy the file system.a from the STANDARD library to a local Domain/Ada library and recompile the file there with the new values.)

The following pragmas are implemented as described in Appendix B of the RM:

- **pragma** ELABORATE
- **pragma** LIST
- **pragma** PAGE
- **pragma** PRIORITY

F.1.2 Implementation-Defined Attribute: 'REF

Domain/Ada provides one implementation-defined attribute, **'REF**. You can use this attribute in one of two ways: *X*'**REF** and **SYSTEM.ADDRESS'REF**(*N*). You can use *X*'**REF** only in machine code procedures while you can use **SYSTEM.ADDRESS'REF**(*N*) anywhere that you want to convert an integer expression to an address.

F.1.2.1 *X*'REF

The *X*'**REF** attribute generates a reference to the entity to which it is applied.

In *X*'**REF**, *X* must be either a constant, variable, procedure, function, or label. The attribute returns a value of the type MACHINE_CODE.OPERAND, which you can use only to designate an operand within a machine code-statement.

You can precede the instruction generated by the code-statement in which the attribute occurs by additional instructions needed to facilitate the reference (for example, loading a base register). If the declarative section of the procedure contains **pragma** IMPLICIT_CODE (OFF), the compiler will generate a warning if additional code is required.

References can also cause the generation of run-time checks. You can use **pragma** SUPPRESS to eliminate these checks.

```
CODE_1'(JSR, PROC'REF);
CODE_2'(MOVE_L, X.ALL(Z)'REF, D0);
```

For more information on machine code insertions, refer to Chapter 9.

F.1.2.2 SYSTEM.ADDRESS'REF(*N*)

The effect of **SYSTEM.ADDRESS'REF(*N*)** is similar to the effect of an unchecked conversion from integer to address. However, you should use this attribute instead of an unchecked conversion in the following circumstances (in these circumstances, *N* must be static):

- Within any of the run-time configuration packages:
  Use of unchecked conversion within an address clause would require the generation of elaboration code, but the configuration packages are not elaborated.

- In any instance where N is greater than INTEGER'LAST:
  Such values are required in address clauses that reference the upper portion of memory. To use unchecked conversion in these instances would require that the expression be given as a negative integer.

- To place an object at an address, use the **'REF** attribute:
  The *integer_value*, in the following example, is converted to an address for use in the address clause representation specification. The form avoids UNCHECKED_CONVERSION and is also useful for 32-bit unsigned addresses.

```
--place an object at an address
for object use at ADDRESS'REF (integer_value)

--to use unsigned addresses
for VECTOR use at SYSTEM.ADDRESS'REF(16#8080000D0#);
TOP_OF_MEMORY: SYSTEM.ADDRESS:= SYSTEM.ADDRESS'REF(16#FFFFFFFF#);
```

In **SYSTEM.ADDRESS'REF(*N*)**, **SYSTEM.ADDRESS** must be the type SYSTEM.ADDRESS. *N* must be an expression of type UNIVERSAL_INTEGER. The attribute returns a value of type SYSTEM.ADDRESS, which represents the address designated by *N*.

F.2 Specification of the Package SYSTEM

```
with unsigned_types; package SYSTEM is

    pragma suppress(ALL_CHECKS);
    pragma suppress(EXCEPTION_TABLES);
    pragma not_elaborated;

        type NAME is ( APOLLO_CROSS_68000 );

        SYSTEM_NAME           : constant NAME := APOLLO_CROSS_68000;

        STORAGE_UNIT : constant := 8;
        MEMORY_SIZE           : constant := 16_777_216;

        -- System-Dependent Named Numbers

        MIN_INT                          : constant := -2_147_483_648;
        MAX_INT                          : constant := 2_147_483_647;
        MAX_DIGITS            : constant := 15;
        MAX_MANTISSA : constant := 31;
        FINE_DELTA            : constant := 2.0**(-31);
        TICK                             : constant := 0.01;

        -- Other System-dependent Declarations

        subtype PRIORITY is INTEGER range 0 .. 99;

        MAX_REC_SIZE : integer := 1024;

        type ADDRESS is private;

        function ">" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "<" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function ">="(A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "<="(A: ADDRESS; B: ADDRESS) return BOOLEAN;
        function "-" (A: ADDRESS; B: ADDRESS) return INTEGER;
        function "+" (A: ADDRESS; I: INTEGER) return ADDRESS;
        function "-" (A: ADDRESS; I: INTEGER) return ADDRESS;

        function "+" (I: UNSIGNED_TYPES.UNSIGNED_INTEGER) return ADDRESS;

        function MEMORY_ADDRESS
                    (I: UNSIGNED_TYPES.UNSIGNED_INTEGER) return ADDRESS renames "+";

        NO_ADDR : constant ADDRESS;

private
```

```
        type ADDRESS is new UNSIGNED_TYPES.UNSIGNED_INTEGER;

        NO_ADDR : constant ADDRESS := 0;

        pragma BUILT_IN(">");
        pragma BUILT_IN("<");
        pragma BUILT_IN(">=");
        pragma BUILT_IN("<=");
        pragma BUILT_IN("-");
        pragma BUILT_IN("+");

end SYSTEM;
```

F.3 Restrictions on Representation Clauses and Unchecked Type Conversions

This section summarizes the restrictions on representation clauses and unchecked type conversions for Domain/Ada.

We describe the representation clauses that Domain/Ada supports in Chapter 11.

F.3.1 Representation Clauses

Domain/Ada supports bit level, length, enumeration, size, and record representation clauses. Size clauses are not supported for tasks, floating-point types, access types, or array types. This implementation supports address clauses for objects except for task objects and for initialized objects given dynamic addresses. Address clauses for task entries are supported; the specified value is a UNIX signal value.

The only restrictions on record representation clauses are the following:

- If a component does not start and end on a storage unit boundary, it must be possible to get the component into a register with one move instruction. On an MC68000 machine, where longwords start on even bytes, the component must fit into 4 bytes starting on a word boundary.
- A component that is itself a record must occupy a power of 2 bits. Components that are of a discrete type or packed array can occupy an arbitrary number of bits subject to the previously mentioned restrictions.

F.3.2 Unchecked Type Conversions

This implementation supports the generic function UNCHECKED_CONVERSION with the following restriction:

- You cannot instantiate the predefined generic function UNCHECKED_CONVERSION with a target type that is an unconstrained array type or an unconstrained record type with discriminants.

F.4 Naming Conventions for Denoting Implementation-Dependent Components in Record Representation Clauses

Record representation clauses are based on the target machine's word, byte, and bit order numbering so that Domain/Ada is consistent with various machine architecture manuals. Bits within a STORAGE_UNIT are also numbered according to the target machine manuals. This implementation of does not support the allocation of implementation-dependent components in records.

F.5 Interpretations of Expressions in Address Clauses

This implementation supports the SYSTEM.ADDRESS'REF(*N*) summarized in Section F.1.2.2.

F.6 Implementation-Dependent Characteristics of I/O Packages

Although not required for validation, Domain/Ada implements all of TEXT_IO, SEQUENTIAL_IO, and DIRECT_IO, allowing programs using a target processor access to the host file system. This support allows transparent access to host file system features and permits easy debugging of embedded system code. The cross I/O system is implemented by having a program on the host computer that receives and executes file system requests issued by the target, and also by having low-level code on the target that sends file system requests to the host when the I/O subprograms listed above are called. Both **a.run** and **a.db** support cross I/O from the target.

To use any of the cross I/O packages you must add the library CROSS_IO to the library search list.

F.6.1 Instantiations of DIRECT_IO

Instantiations of DIRECT_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as a string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. You can change MAX_REC_SIZE (defined in package SYSTEM) before instantiating DIRECT_IO to provide an upper limit on the record size. The maximum size supported is 1024 * 1024 * STORAGE_UNIT bits. DIRECT_IO will raise USE_ERROR if MAX_REC_SIZE exceeds this absolute limit.

F.6.2 Instantiations of SEQUENTIAL_IO

Instantiations of SEQUENTIAL_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as STRING where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. You can change MAX_REC_SIZE (defined in **package** SYSTEM) before instantiating INTEGER_IO to provide an upper limit on the record size. SEQUENTIAL_IO imposes no limit on MAX_REC_SIZE.

F.7 Additional Implementation-Dependent Features

This section details any other features that are specific to this implementation.

F.7.1 Restrictions on 'Main' Programs

Domain/Ada requires that a 'main' program must be a non-generic subprogram that is either a procedure or a function returning an Ada STANDARD.INTEGER (the predefined type). In addition, a 'main' program cannot be an instantiation of a generic subprogram.

F.7.2 Generic Declarations

Domain/Ada does not require that a generic declaration and the corresponding body be part of the same compilation, and they are not required to exist in the same Domain/Ada library. The compiler generates an error if a single compilation contains two versions of the same unit.

F.7.3 Implementation-Dependent Portions of Predefined Ada Packages

Domain/Ada supplies the following predefined Ada packages given by the Ada RM C(22) in the **standard** and **cross_io** libraries:

- **package** STANDARD
- **package** CALENDAR
- **package** SYSTEM
- **generic procedure** UNCHECKED_DEALLOCATION
- **generic function** UNCHECKED_CONVERSION
- **generic package** SEQUENTIAL_IO
- **generic package** DIRECT_IO

- **package** TEXT_IO
- **package** IO_EXCEPTIONS
- **package** LOW_LEVEL_IO
- **package** MACHINE_CODE

F.7.4 Values of Integer Attributes

Domain/Ada provides three integer types in addition to *universal_integer*: INTEGER, SHORT_INTEGER, and TINY_INTEGER. Table F-1 lists the ranges for these integer types.

Table F-1. Domain/Ada Integer Types

Name of Attribute	Attribute Value of INTEGER	Attribute Value of SHORT_INTEGER	Attribute Value of TINY_INTEGER
SIZE	32	16	8
FIRST	-2_147_483_648	-32_768	-128
LAST	2_147_483_647	32_767	127

F.7.5 Values of Floating-Point Attributes

Table F-2 lists the attributes of floating-point types.

Table F-2. Domain/Ada Floating-Point Types

Name of Attribute	Attribute Value of FLOAT	Attribute Value of SHORT_FLOAT
SIZE	64	32
FIRST	-1.79769313486232E+308	-3.40282E+38
LAST	1.79769313486232E+308	3.40282E+38
DIGITS	15	6
MANTISSA	51	21
EPSILON	8.88178419700125E-16	9.53674316406250E-07
EMAX	204	84
SMALL	1.94469227433161E-62	2.58493941422821E-26
LARGE	2.57110087081438E+61	1.93428038904620E+25
SAFE_EMAX	1022	126
SAFE_SMALL	1.11253692925360E-308	5.87747175411144E-39
SAFE_LARGE	4.49423283715579E+307	8.5075511654154E+37
MACHINE_RADIX	2	2
MACHINE_MANTISSA	53	24
MACHINE_EMAX	1024	128
MACHINE_EMIN	-1022	-126
MACHINE_ROUNDS	TRUE	TRUE
MACHINE_OVERFLOWS	TRUE	TRUE

F.7.6 Attributes of Type DURATION

Table F-3 lists the attributes for the fixed-point type DURATION.

Table F-3. Attributes for the Fixed-Point Type DURATION

Name of Attribute	Attribute Value for DURATION
SIZE	32
FIRST LAST	-2147483.648 2147483.647
DELTA	1.00000000000000E-03
MANTISSA	31
SMALL LARGE	1.00000000000000E-03 2.14748364700000E+06
FORE AFT	8 3
SAFE_SMALL SAFE_LARGE	1.00000000000000E-03 2.14748364700000E+06
MACHINE_ROUNDS MACHINE_OVERFLOWS	TRUE TRUE

F.7.7 Implementation Limits

Character Set: Domain/Ada provides the full *graphic_character* textual representation for programs. The character set for source files and internal character representations is ASCII.

Lexical Elements, Separators, and Delimiters: Domain/Ada uses normal Domain/OS I/O text files as input. Each line is terminated by a newline character (ASCII.LF).

Source File Limits: Domain/Ada imposes the following limitations on source files:

- 499 characters per source line
- 1296 Ada units per source file
- 32767 lines per source file

Compiler/Tool Limits: Domain/Ada imposes the following limits on the use of the Domain/Ada compiler:

- 499 characters in identifiers and literals
- 4,000,000 STORAGE UNITS in a statically sized record or array
- 10,240 bytes as the storage size default for a task (if tasks need larger or smaller stack sizes, the 'STORAGE_SIZE attribute may be used with the task type declaration)
- 400 bytes as the STORAGE_SIZE default collection size for access type
- No limit on the number of declared objects (except virtual space)
- 800 characters in a rooted name (full pathname of an object)
- 8 recursive inlines
- 8 nested inlines
- 400 nested constructs
- 2048 characters in ADAPATH (library search list)
- 2048 characters in a WITH or INFO directive
- 16M of memory use per compilation (other Domain/OS limits may apply)
- 50 lexical errors before the front end exits
- 100 syntax errors before the front end exits
- 10 attempts to lock GVAS_table
- 10 attempts to lock **ada.lib**
- 20 attempts to lock **gnrx.lib**
- 64 debugger breakpoints
- 32 debugger array dimensions in a **p** command
- 9 debugger 'call parameters'
- 256 debugger 'run parameters'

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

```
-- MACRO.DFS

-- THIS FILE CONTAINS THE MACRO DEFINITIONS USED IN THE ACVC TESTS.
-- THESE DEFINITIONS ARE USED BY THE ACVC TEST PRE-PROCESSOR,
-- MACROSUB. MACROSUB WILL CALCULATE VALUES FOR THOSE MACRO SYMBOLS
-- WHOSE DEFINITIONS DEPEND ON THE VALUE OF MAX_IN_LEN (NAMELY, THE
-- VALUES OF THE MACRO SYMBOLS BIG_ID1, BIG_ID2, BIG_ID3, BIG_ID4,
-- BIG_STRING1, BIG_STRING2, MAX_STRING_LITERAL, BIG_INT_LIT,
-- BIG_REAL_LIT, MAX_LEN_INT_BASED_LITERAL, MAX_LEN_REAL_BASED_LITERAL,
-- AND BLANKS).  THEREFORE, ANY VALUES GIVEN IN THIS FILE FOR THOSE
-- MACRO SYMBOLS WILL BE IGNORED BY MACROSUB.

-- NOTE: THE MACROSUB PROGRAM EXPECTS THE FIRST MACRO IN THIS FILE TO
--       BE MAX_IN_LEN.

-- EACH DEFINITION IS ACCORDING TO THE FOLLOWING FORMAT:

-- A. A NUMBER OF LINES PRECEDED BY THE ADA COMMENT DELIMITER, --.
--    THE FIRST OF THESE LINES CONTAINS THE MACRO SYMBOL AS IT APPEARS
--    IN THE TEST FILES (WITH THE DOLLAR SIGN).  THE NEXT FEW "COMMENT"
--    LINES CONTAIN A DESCRIPTION OF THE VALUE TO BE SUBSTITUTED.
--    THE REMAINING "COMMENT" LINES, THE FIRST OF WHICH BEGINS WITH THE
--    WORDS "USED IN: " (NO QUOTES), CONTAIN A LIST OF THE TEST FILES
--    (WITHOUT THE .TST EXTENSION) IN WHICH THE MACRO SYMBOL APPEARS.
--    EACH TEST FILE NAME IS PRECEDED BY ONE OR MORE BLANKS.
-- B. THE IDENTIFIER (WITHOUT THE DOLLAR SIGN) OF THE MACRO SYMBOL,
--    FOLLOWED BY A SPACE OR TAB, FOLLOWED BY THE VALUE TO BE
--    SUBSTITUTED.  IN THE DISTRIBUTION FILE, A SAMPLE VALUE IS
--    PROVIDED; THIS VALUE MUST BE REPLACED BY A VALUE APPROPRIATE TO
--    THE IMPLEMENTATION.

-- DEFINITIONS ARE SEPARATED BY ONE OR MORE EMPTY LINES.
-- THE LIST OF DEFINITIONS BEGINS AFTER THE FOLLOWING EMPTY LINE.

-- $MAX_IN_LEN
-- AN INTEGER LITERAL GIVING THE MAXIMUM LENGTH PERMITTED BY THE
-- COMPILER FOR A LINE OF ADA SOURCE CODE (NOT INCLUDING AN END-OF-LINE
-- CHARACTER).
-- USED IN:  A26007A
MAX_IN_LEN    499

-- $BIG_ID1
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  B23003D  B23003E  C23003G
--           C23003H  C23003I  C23003J  C35502D  C35502F
BIG_ID1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1

-- $BIG_ID2
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID1 ONLY IN THE LAST CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  B23003F  C23003G  C23003H
--           C23003I  C23003J
BIG_ID2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2

-- $BIG_ID3
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID3
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA3AAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

-- $BIG_ID4
-- AN IDENTIFIER IN WHICH THE NUMBER OF CHARACTERS IS $MAX_IN_LEN,
-- DIFFERING FROM $BIG_ID3 ONLY IN THE MIDDLE CHARACTER.
-- USED IN:  C23003A  C23003B  C23003C  C23003G  C23003H  C23003I
--           C23003J
BIG_ID4
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA4AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

-- $BIG_STRING1
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING2
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA"

-- $BIG_STRING2
-- A STRING LITERAL (WITH QUOTES) WHOSE CATENATION WITH $BIG_STRING1
-- ($BIG_STRING1 & $BIG_STRING2) PRODUCES THE IMAGE OF $BIG_ID1.
-- USED IN:  C35502D  C35502F
BIG_STRING2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA1"

-- $MAX_STRING_LITERAL
-- A STRING LITERAL CONSISTING OF $MAX_IN_LEN CHARACTERS (INCLUDING THE
-- QUOTE CHARACTERS).
-- USED IN:  A26007A
MAX_STRING_LITERAL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

-- $NEG_BASED_INT
-- A BASED_INTEGER LITERAL (PREFERABLY BASE 8 OR 16) WHOSE HIGHEST ORDER
-- NON-ZERO BIT WOULD FALL IN THE SIGN BIT POSITION OF THE
-- REPRESENTATION FOR SYSTEM.MAX_INT, I.E., AN ATTEMPT TO WRITE A
-- NEGATIVE VALUED LITERAL SUCH AS -2 BY TAKING ADVANTAGE OF THE
-- BIT REPRESENTATION.
-- USED IN:  E24201A
NEG_BASED_INT    16#FFFFFFFD#

$BIG_INT_LIT
-- AN INTEGER LITERAL WHOSE VALUE IS 298, BUT WHICH HAS
-- ($MAX_IN_LEN - 3) LEADING ZEROES.
-- USED IN:  C24003A
BIG_INT_LIT
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000298

$BIG_REAL_LIT
-- A UNIVERSAL_REAL LITERAL WHOSE VALUE IS 690.0, BUT WHICH HAS
-- ($MAX_IN_LEN - 5) LEADING ZEROES.
-- USED IN:  C24003B  C24003C
BIG_REAL_LIT
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
00000000000000000000000000000069.0E1

-- $MAX_LEN_INT_BASED_LITERAL
-- A BASED INTEGER LITERAL (USING COLONS) WHOSE VALUE IS 2:11:, HAVING
-- ($MAX_IN_LEN - 5) ZEROES BETWEEN THE FIRST COLON AND THE FIRST 1.
-- USED IN:  C2A009A
MAX_LEN_INT_BASED_LITERAL
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000011:

-- $MAX_LEN_REAL_BASED_LITERAL
-- A BASED REAL LITERAL (USING COLONS) WHOSE VALUE IS 16:F.E:, HAVING
-- ($MAX_IN_LEN - 7) ZEROES BETWEEN THE FIRST COLON AND THE F.
-- USED IN:  C2A009A
MAX_LEN_REAL_BASED_LITERAL
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000F.E:

-- $BLANKS
-- A SEQUENCE OF ($MAX_IN_LEN - 20) BLANKS.
-- USED IN:  B22001A  B22001B  B22001C  B22001D  B22001E  B22001F
--           B22001G  B22001I  B22001J  B22001K  B22001L  B22001M
--           B22001N
--      <       LIMITS OF SAMPLE SHOWN BY ANGLE BRACKETS          >
BLANKS




-- $MAX_DIGITS
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_DIGITS.
-- USED IN:  B35701A  CD7102B
MAX_DIGITS    15

-- $NAME
-- THE NAME OF A PREDEFINED INTEGER TYPE OTHER THAN INTEGER,
-- SHORT_INTEGER, OR LONG_INTEGER.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS   NO_SUCH_TYPE_AVAILABLE.)
-- USED IN:  AVAT007  C45231D  B86001X
NAME    TINY_INTEGER

-- $FLOAT_NAME
-- THE NAME OF A PREDEFINED FLOATING POINT TYPE OTHER THAN FLOAT,
-- SHORT_FLOAT, OR LONG_FLOAT.  (IMPLEMENTATIONS WHICH HAVE NO SUCH
-- TYPES SHOULD USE AN UNDEFINED IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT013  B86001Y
FLOAT_NAME    NO_SUCH_FIXED_TYPE

-- $FIXED_NAME
-- THE NAME OF A PREDEFINED FIXED POINT TYPE OTHER THAN DURATION.
-- (IMPLEMENTATIONS WHICH HAVE NO SUCH TYPES SHOULD USE AN UNDEFINED
-- IDENTIFIER SUCH AS NO_SUCH_TYPE.)
-- USED IN:  AVAT030  B86001Z
FIXED_NAME    NO_SUCH_TYPE

-- $INTEGER_FIRST
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS INTEGER'FIRST.
-- USED IN:  C35503F  B54B01B
INTEGER_FIRST    -2147483648

-- $INTEGER_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST.
-- USED IN:  C35503F  B45232A  B45B01B
```

```
INTEGER_LAST    2147483647

-- $INTEGER_LAST_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS INTEGER'LAST + 1.
-- USED IN:  C45232A
INTEGER_LAST_PLUS_1    2_147_483_648

-- $MIN_INT
-- AN INTEGER LITERAL, WITH SIGN, WHOSE VALUE IS SYSTEM.MIN_INT.
-- THE LITERAL MUST NOT CONTAIN UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  CD7101B
MIN_INT    -2147483648

-- $MAX_INT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT.
-- THE LITERAL MUST NOT INCLUDE UNDERSCORES OR LEADING OR TRAILING
-- BLANKS.
-- USED IN:  C35503D  C35503F  C4A007A  CD7101B
MAX_INT    2147483647

-- $MAX_INT_PLUS_1
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_INT + 1.
-- USED IN:  C45232A
MAX_INT_PLUS_1    2_147_483_648

-- $LESS_THAN_DURATION
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE (NOT SUBJECT TO
-- ROUND-OFF ERROR IF POSSIBLE) LIES BETWEEN DURATION'BASE'FIRST AND
-- DURATION'FIRST.  IF NO SUCH VALUES EXIST, USE A VALUE IN
-- DURATION'RANGE.
-- USED IN:  C96005B
LESS_THAN_DURATION    -100_000.0

-- $GREATER_THAN_DURATION
-- A REAL LITERAL WHOSE VALUE (NOT SUBJECT TO ROUND-OFF ERROR
-- IF POSSIBLE) LIES BETWEEN DURATION'BASE'LAST AND DURATION'LAST.  IF
-- NO SUCH VALUES EXIST, USE A VALUE IN DURATION'RANGE.
-- USED IN:  C96005B
GREATER_THAN_DURATION    100_000.0

-- $LESS_THAN_DURATION_BASE_FIRST
-- A REAL LITERAL (WITH SIGN) WHOSE VALUE IS LESS THAN
-- DURATION'BASE'FIRST.
-- USED IN:  C96005C
LESS_THAN_DURATION_BASE_FIRST    -10_000_000.0

-- $GREATER_THAN_DURATION_BASE_LAST
-- A REAL LITERAL WHOSE VALUE IS GREATER THAN DURATION'BASE'LAST.
-- USED IN: C96005C
GREATER_THAN_DURATION_BASE_LAST    10_000_000.0

-- $COUNT_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.COUNT'LAST.
-- USED IN:  CE3002B
COUNT_LAST    2_147_483_647

-- $FIELD_LAST
-- AN INTEGER LITERAL WHOSE VALUE IS TEXT_IO.FIELD'LAST.
-- USED IN:  CE3002C
FIELD_LAST    2_147_483_647

-- $ILLEGAL_EXTERNAL_FILE_NAME1
-- AN ILLEGAL EXTERNAL FILE NAME (E.G., TOO LONG, CONTAINING INVALID
-- CHARACTERS, CONTAINING WILD-CARD CHARACTERS, OR SPECIFYING A
-- NONEXISTENT DIRECTORY).
-- USED IN:  CE2103A  CE2102C  CE2102H  CE2103B  CE3102B  CE3107A
ILLEGAL_EXTERNAL_FILE_NAME1    /illegal/file_name/2{}$%2102C.DAT

-- $ILLEGAL_EXTERNAL_FILE_NAME2
-- AN ILLEGAL EXTERNAL FILE NAME, DIFFERENT FROM $EXTERNAL_FILE_NAME1.
-- USED IN:  CE2102C  CE2102H  CE2103A  CE2103B
```

```
ILLEGAL_EXTERNAL_FILE_NAME2    /illegal/file_name/CE2102C*.DAT

-- $ACC_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE MINIMUM NUMBER OF BITS
-- SUFFICIENT TO HOLD ANY VALUE OF AN ACCESS TYPE.
-- USED IN:  CD1C03C  CD2A81A  CD2A81B  CD2A81C  CD2A81D  CD2A81E
--           CD2A81F  CD2A81G  CD2A83A  CD2A83B  CD2A83C  CD2A83E
--           CD2A83F  CD2A83G  ED2A86A  CD2A87A
ACC_SIZE    32

-- $TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS A SINGLE ENTRY WITH ONE INOUT PARAMETER.
-- USED IN:  CD2A91A  CD2A91B  CD2A91C  CD2A91D  CD2A91E
TASK_SIZE    32

-- $MIN_TASK_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS THE NUMBER OF BITS REQUIRED TO
-- HOLD A TASK OBJECT WHICH HAS NO ENTRIES, NO DECLARATIONS, AND "NULL;"
-- AS THE ONLY STATEMENT IN ITS BODY.
-- USED IN:  CD2A95A
MIN_TASK_SIZE    32

-- $NAME_LIST
-- A LIST OF THE ENUMERATION LITERALS IN THE TYPE SYSTEM.NAME, SEPARATED
-- BY COMMAS.
-- USED IN:  CD7003A
-- .XXXX change to your system name
NAME_LIST    APOLLO_CROSS_68000

-- $DEFAULT_SYS_NAME
-- THE VALUE OF THE CONSTANT SYSTEM.SYSTEM_NAME.
-- USED IN:  CD7004A  CD7004C  CD7004D
-- .XXXX change to your system name
DEFAULT_SYS_NAME    APOLLO_CROSS_68000

-- $NEW_SYS_NAME
-- A VALUE OF THE TYPE SYSTEM.NAME, OTHER THAN $DEFAULT_SYS_NAME.  IF
-- THERE IS ONLY ONE VALUE OF THE TYPE, THEN USE THAT VALUE.
-- NOTE: IF THERE ARE MORE THAN TWO VALUES OF THE TYPE, THEN THE
--       PERTINENT TESTS ARE TO BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7004B1
-- .XXXX change to your system name
NEW_SYS_NAME    APOLLO_CROSS_68000

-- $DEFAULT_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.STORAGE_UNIT.
-- USED IN:  CD7005B  ED7005D3M  CD7005E
DEFAULT_STOR_UNIT    8

-- $NEW_STOR_UNIT
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA STORAGE_UNIT, OTHER THAN $DEFAULT_STOR_UNIT.  IF THERE
-- IS NO OTHER PERMITTED VALUE, THEN USE THE VALUE OF
-- $SYSTEM.STORAGE_UNIT.  IF THERE IS MORE THAN ONE ALTERNATIVE,
-- THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR EACH ALTERNATIVE.
-- USED IN:  ED7005C1  ED7005D1  CD7005E
NEW_STOR_UNIT    8

-- $DEFAULT_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MEMORY_SIZE.
-- USED IN:  CD7006B  ED7006D3M  CD7006E
-- .XXXX insert your system.memory_size
DEFAULT_MEM_SIZE    16_777_216

-- $NEW_MEM_SIZE
-- AN INTEGER LITERAL WHOSE VALUE IS A PERMITTED ARGUMENT FOR
-- PRAGMA MEMORY_SIZE, OTHER THAN $DEFAULT_MEM_SIZE.  IF THERE IS NO
-- OTHER VALUE, THEN USE $DEFAULT_MEM_SIZE.  IF THERE IS MORE THAN
-- ONE ALTERNATIVE, THEN THE PERTINENT TESTS SHOULD BE RUN ONCE FOR
-- EACH ALTERNATIVE.  IF THE NUMBER OF PERMITTED VALUES IS LARGE, THEN
-- SEVERAL VALUES SHOULD BE USED, COVERING A WIDE RANGE OF
```

```
-- POSSIBILITIES.
-- USED IN:  ED7006C1  ED7006D1  CD7006E
NEW_MEM_SIZE     16_777_216

-- $LOW_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE LOWER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
LOW_PRIORITY     0

-- $HIGH_PRIORITY
-- AN INTEGER LITERAL WHOSE VALUE IS THE UPPER BOUND OF THE RANGE
-- FOR THE SUBTYPE SYSTEM.PRIORITY.
-- USED IN:  CD7007C
HIGH_PRIORITY     99

-- $MANTISSA_DOC
-- AN INTEGER LITERAL WHOSE VALUE IS SYSTEM.MAX_MANTISSA AS SPECIFIED
-- IN THE IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013B
MANTISSA_DOC     31

-- $DELTA_DOC
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.FINE_DELTA AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7013D
DELTA_DOC     0.00000000046566128730773925781 25

-- $TICK
-- A REAL LITERAL WHOSE VALUE IS SYSTEM.TICK AS SPECIFIED IN THE
-- IMPLEMENTOR'S DOCUMENTATION.
-- USED IN:  CD7104B
TICK     0.01
```

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G
This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E
This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A
This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B
This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D
This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B
This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B
This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A
This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D
This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I
This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C
This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A
This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B
This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C
This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

Apollo Computer Inc.

Compiler:	Domain/ADA, Ver 3.0.MBX
ACVC Version:	1.10

Attachment II

Invoking the Domain/Ada Compiler: Syntax and Options

SYNTAX

ada [*options*] [*source_file*] . . . [*linker_options*] [*object_file*.**o**] . . .

OPTIONS

-a *file_name*	treat file as an **ar** library.
-d	analyze for dependencies only.
-e	process compilation error messages using **a.error** and direct the output to **stdout**. Only source lines containing errors are listed. Use only one **-e** or **-E** option.
-E	
-E *file*	
-E *directory*	without a file or directory argument, **ada** processes error messages using **a.error** and directs a brief output to **stdout**; the raw error messages are left in *ada_source*.**err**. If a file pathname is given, the raw error messages are placed in that file. If a directory argument is supplied, the raw error output is placed in *dir/source*.**err**. The file of raw error messages can be used as input to **a.error**. Use only one **-e** or **-E** option.
-el	intersperse error messages among source lines and direct to stdout.
-El	
-El *file*	
-El *directory*	same as the **-E** option, except that a source listing file interspersed with errors is produced.
-ev	process raw error messages using **a.error** and call the environment editor, EDITOR on the source file. If EDITOR is undefined, invoke **vi**.
-K	keep the intermediate language (IL) file produced by the compiler front end.
-l*file_abbreviation*	link this library file (no space after **-l**).
-M *unit_name*	
-M *ada_source.a*	produce an executable program using the named unit or source root name as the main program.
-o *executable_file*	set output to *executable_file*; default is **a.vox**.
-O[0-9]	invoke the code optimizer (no space before the digit); an optional digit limits the number of passes by the optimizer; without the **-O** option, one pass is

made (default); **-O0** prevents optimization; **-O** with no digit specifies maximum optimization.

-R *library* — force analysis of all generic instantiations, causing reinstantiation of any that are out of date.

-S — apply **pragma** SUPPRESS to the compilation.

-sh — display the name of the tool executable but do not execute it.

-T — print timing information for the compilation.

-v — print additional information about the compilation.

-w — suppress warning diagnostics.